CS333 APPLICATION SOFTWARE DEVELOPMENT LAB

MySql and PHP

Manual

September 12, 2018

- **Course Objectives**

  1. To introduce basic commands and operations on database.
  2. To introduce stored programming concepts (PL-SQL) using Cursors and Triggers .
  3. To familiarize front end tools of database.

- **Expected Outcome**

  The students will be able to

  1. Design and implement a database for a given proble

     m using database design principles.
  2. Apply stored programming concepts (PL-SQL) using Cursors and Triggers.
  3. Use graphical user interface, Event Handling and Database connectivity to develop and deploy applications and applets.
  4. Develop medium-sized project in a team.

Contents

Chapter 1

Introduction

Oracle database (Oracle DB) is a relational database management system (RDBMS) from the Oracle Corporation. Originally developed in 1977 by Lawrence Ellison and other developers, Oracle DB is one of the most trusted and widely-used relational database engines.

The system is built around a relational database framework in which data objects may be directly accessed by users (or an application front end) through structured query language (SQL). Oracle is a fully scalable relational database architecture and is often used by global enterprises, which manage and process data across wide and local area networks. The Oracle database has its own network component to allow communications across networks.

Oracle DB is also known as Oracle RDBMS and, sometimes, just Oracle. Every organization has information that it must store and manage to meet its requirements. For example, a corporation must collect and maintain human resources records for its employees. This information must be available to those who need it.

An information system is a formal system for storing and processing information. An information system could be a set of cardboard boxes containing manila folders along with rules for how to store and retrieve the folders. However, most companies today use a database to automate their information systems. A database is an organized collection of information treated as a unit. The purpose of a database is to collect, store, and retrieve related information for use by database applications.

Database Management System (DBMS)

A database management system (DBMS) is a software that controls the storage, organization, and retrieval of data.
Typically, a DBMS has the following elements:

1. Kernel code

This code manages memory and storage for the DBMS.

2. Repository of metadata

   This repository is usually called a data dictionary.

3. Query language

   This language enables applications to access the data.

A database application is a software program that interacts with a database to access and manipulate data.The first generation of database management systems included the following types:

1. Hierarchical

   A hierarchical database organizes data in a tree structure. Each parent record has one or more child records, similar to the structure of a file system.

2. Network

   A network database is similar to a hierarchical database, except records have a many-to-many rather than a one-to-many relationship.

The preceding database management systems stored data in rigid, predetermined relationships. Because no data definition language existed, changing the structure of the data was difficult. Also, these systems lacked a simple query language, which hindered application development.

Relational Model

A relational database is a database that conforms to the relational model. The relational model has the following major aspects:

1. Structures

   Well-defined objects store or access the data of a database.

2. Operations

   Clearly defined actions enable applications to manipulate the data and structures of a database.

3. Integrity rules

   Integrity rules govern operations on the data and structures of a database.

A relational database stores data in a set of simple relations. A relation is a set of tuples. A tuple is an unordered set of attribute values.

A table is a two-dimensional representation of a relation in the form of rows (tuples) and columns (attributes). Each row in a table has the same set of columns. A relational database is a database that stores data in relations (tables). For example, a relational database could store information about company employees in an employee table, a department table, and a salary table.

Relational Database Management System (RDBMS)

The relational model is the basis for a relational database management system (RDBMS). An RDBMS moves data into a database, stores the data, and retrieves it so that applications can manipulate it.

Oracle Database is an RDBMS. An RDBMS that implements object-oriented features such as user-defined types, inheritance, and polymorphism is called an object-relational database management system (ORDBMS). Oracle Database has extended the relational model to an object-relational model, making it possible to store complex business models in a relational database.

Brief History of Oracle Database

The current version of Oracle Database is the result of over 35 years of innovative development.

Highlights in the evolution of Oracle Database include the following:

1. Founding of Oracle

   In 1977, Larry Ellison, Bob Miner, and Ed Oates started the consultancy Software Development Laboratories, which became Relational Software, Inc. (RSI). In 1983, RSI became Oracle Systems Corporation and then later Oracle Corporation.

2. First commercially available RDBMS

   In 1979, RSI introduced Oracle V2 (Version 2) as the first commercially available SQL-based RDBMS, a landmark event in the history of relational databases.

3. Portable version of Oracle Database

Oracle Version 3, released in 1983, was the first relational database to run on mainframes, minicomputers, and PCs. The database was written in C, enabling the database to be ported to multiple platforms.

4. Enhancements to concurrency control, data distribution, and scalability

   Version 4 introduced multiversion read consistency. Version 5, released in 1985, supported client/server computing and distributed database systems. Version 6 brought enhancements to disk I/O, row locking, scalability, and backup and recovery. Also, Version 6 introduced the first version of the PL/SQL language, a proprietary procedural extension to SQL.

5. PL/SQL stored program units

   Oracle7, released in 1992, introduced PL/SQL stored procedures and triggers.

6. Objects and partitioning

   Oracle8 was released in 1997 as the object-relational database, supporting many new data types. Additionally, Oracle8 supported partitioning of large tables.

7. Internet computing

   Oracle8i Database, released in 1999, provided native support for internet protocols and server-side support for Java. Oracle8i was designed for internet computing, enabling the database to be deployed in a multitier environment.

8. Oracle Real Application Clusters (Oracle RAC)

   Oracle9i Database introduced Oracle RAC in 2001, enabling multiple instances to access a single database simultaneously. Additionally, Oracle XML Database (Oracle XML DB) introduced the ability to store and query XML.

9. Grid computing

   Oracle Database 10g introduced grid computing in 2003. This release enabled organizations to virtualize computing resources by building a grid infrastructure based on low-cost commodity servers. A key goal was to make the database self-managing and self-tuning. Oracle Automatic Storage Management (Oracle ASM) helped achieve this goal by virtualizing and simplifying database storage management.

10. Manageability, diagnosability, and availability

    Oracle Database 11g, released in 2007, introduced a host of new features that enabled administrators and developers to adapt quickly to changing business requirements. The key to adaptability is simplifying the information infrastructure by consolidating information and using automation wherever possible.

11. Plugging In to the Cloud

    Oracle Database 12c, released in 2013, was designed for the Cloud, featuring a new Multitenant architecture, In-Memory column store, and support for JSON documents. Oracle Database 12c helps customers make more efficient use of their IT resources, while continuing to reduce costs and improve service levels for users.

Chapter 2

Installing Oracle

Installing Oracle XE on Ubuntu 32-bit
The Link : http://1drv.ms/1vMWia6 Note:- For successful installation, following requirements must be satisfied :-
a) 512 MB RAM
b) 1 GB swap-space
c) 1.5 GB of free space in the mount-point containing /usr/lib directory.

For installing the Oracle 10g XE R2 do the following : 1) Install the libaio1 package . sudo apt-get install libaio1

2) Install the Oracle 10g XE : dpkg -i oracle-xe$_1$0.2.0.1 − 1.0$_i$386.*deb*

3) Configure the installation: /etc/init.d/oracle-xe configure

Enter the following configuration information: A valid HTTP port for the Oracle Application Express (the default is 8080) A valid port for the Oracle database listener (the default is 1521) A password for the SYS and SYSTEM administrative user accounts Confirm password for SYS and SYSTEM administrative user accounts Whether you want the database to start automatically when the computer starts (next reboot).

4) Set-up the environment variables : Add following lines to .bashrc of the user which you will using to access the Oracle 10g XE: export ORACLE$_H$*OME* = */usr/lib/oracle/xe/app/oracle/product*/10.2.0/*serverexportORACLE*$_S$*ID* = *XEexportPATH* =ORACLE$_H$*OME*/*bin* :PATH

5) Start a new shell (terminal window) for the changes to take effect or execute the .profile to force the changes made in the same session. . ./.profile

6) Use the command "sqlplus sys as sysdba" and the password which you created while configuration to connect to the database.

Link : http://1drv.ms/1vMWia6

Chapter 3

Data Types in MySQL

Each column in a database table is required to have a name and a data type.

SQL developers have to decide what types of data will be stored inside each and every table column when creating a SQL table. The data type is a label and a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

The following table lists the general data types in SQL:

Data type	Description
CHARACTER(n)	Character string. Fixed-length n
VARCHAR(n) or CHARACTER VARYING(n)	Character string. Variable length. Maximum length n
BINARY(n)	Binary string. Fixed-length n
BOOLEAN	Stores TRUE or FALSE values
VARBINARY(n) or BINARY VARYING(n)	Binary string. Variable length. Maximum length n
INTEGER(p)	Integer numerical (no decimal). Precision p
SMALLINT	Integer numerical (no decimal). Precision 5
INTEGER	Integer numerical (no decimal). Precision 10
BIGINT	Integer numerical (no decimal). Precision 19
DECIMAL(p,s)	Exact numerical, precision p, scale s. Example: decimal(5,2)is a number that has 3 digits before the decimal and 2 digits after the decimal
NUMERIC(p,s)	Exact numerical, precision p, scale s. (Same as DECIMAL)

FLOAT(p)	Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a single number specifying the minimum precision
REAL	Approximate numerical, mantissa precision 7
FLOAT	Approximate numerical, mantissa precision 16
DOUBLE PRECISION	Approximate numerical, mantissa precision 16
DATE	Stores year, month, and day values
TIME	Stores hour, minute, and second values
TIMESTAMP	Stores year, month, day, hour, minute, and second values
INTERVAL	Composed of a number of integer fields, representing a period of time, depending on the type of interval
ARRAY	A set-length and ordered collection of elements
MULTISET	A variable-length and unordered collection of elements
XML	Stores XML data

Chapter 4

Experiment 1

Experiment: Create an Employee data base with the data given in table .

Name	Age	Salary	Title
Mahesh	50	95000	Manager
Nimitha	35	50000	Programmer
Ashitha	28	35000	Programmer
Ajay	32	27000	Technician
Roy	24	34000	Programmer

Perform the following retrieval operations.

1. Select all employees from your employee table.
2. Select all employees with a salary over 30000.
3. Select employees who are under 30 years old.
4. Select Name and salary of employees with "Programmer" in their title.
5. Select employees whose last name contains "itha".
6. Select the first name for everyone whose name starts with "A".
7. Select all columns for everyone over 35 years old.
8. Find the employee who is a Technician.
9. Find the average salary of all employees
10. Arrange all employees according to their names in the alphabetical order.

11. Select all employees in the increasing order of their age.
12. Select employees with maximum and minimum salary
13. Find number of employees with salary greater than 30000
14. Select all employees with 'h' in the third positions of their names and age between 20 and 50
15. Display the number of employees working in each title.

Learning Objectives:

1. To create a data base
2. To view the databases created
3. To delete the database
4. To create tables in the database for a schema
5. To view tables and schema
6. To insert tuples into the table

Prerequisite Knowledge:

1. **Create a database on the sql server.**
   create database [databasename];
2. **List all databases on the sql server.**
   show databases;
3. **Switch to a database.**
   use [db name];
4. **To see all the tables in the db.**
   show tables;
5. **To see database's field formats.**
   describe [table name];
6. **To delete a db.**
   drop database [database name];

7. **To delete a table.**
   drop table [table name];

8. **Show all data in a table.**
   *SELECT * FROM [table name];*

9. **To Return the columns and column information pertaining to the designated table.**
   show columns from [table name];

10. **To Show certain selected rows with the value "whatever".**
    *SELECT * FROM [table name] WHERE [field name] = "whatever";*

11. **The INSERT INTO statement is used to insert new records in a table.**
    INSERT INTO tablename (column1,column2)VALUES(value1,value2);

    If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. The INSERT INTO syntax would be as follows:

    INSERT INTO tablename VALUES (value1, value2);

12. **The SELECT statement is used to select data from a database.**
    SELECT column1, column2 FROM tablename;

13. **The SELECT DISTINCT statement is used to return only distinct (different) values.**
    SELECT DISTINCT column1, column2 FROM tablename;

14. **The WHERE clause is used to filter records.**
    SELECT column1, column2 FROM tablename WHERE condition;

15. **The WHERE clause**
    The WHERE clause can be combined with AND, OR, and NOT operators when multiple conditions are to be satisfied
    SELECT column FROM tablename WHERE condition1 AND condition2;

16. **The ORDER BY keyword**
    The ORDER BY keyword is used to sort the result-set in ascending or descending order.
    SELECT column1 FROM tablename ORDER BY column1 ASC—DESC;

17. **The SQL MIN() and MAX() Functions**
The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.
SELECT MIN(column)FROM tablename WHERE condition;

18. **The SQL COUNT(), AVG() and SUM() Functions**
The COUNT() function returns the number of rows that matches a specified criteria.

The AVG() function returns the average value of a numeric column.

The SUM() function returns the total sum of a numeric column.
SELECT COUNT(column)FROM tablename WHERE condition;

19. **The SQL LIKE Operator**
The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards used in conjunction with the LIKE operator:

The percent sign represents zero, one, or multiple characters
The underscore represents a single character

SELECT column1 FROM tablename WHERE column LIKE pattern;
Usage of like operator

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%_%'	Finds any values that starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%_%'	Finds any values that starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

20. **SQL IN Operator**
    The IN operator allows you to specify multiple values in a WHERE clause.
    SELECT column FROM tablename WHERE columnname IN (value1,value2);

    SELECT column FROM table WHERE column IN(SELECT STATEMENT);

21. **SQL IN Operator**
    The IN operator allows you to specify multiple values in a WHERE clause.
    SELECT column FROM tablename WHERE columnname IN (value1,value2);

Chapter 5

Experiment 2

Experiment: Create the given table and perform the following function.

Employee Id	Name	Age	Salary	Title	Joining Date
E1	Mahesh	50	115000.89	Manager	20-03-1967
E2	Nimitha	35	110000.78	Programmer	02-08-1981
E3	Ashitha	28	35000.89	Programmer	01-09-1988
E4	Ajay	32	27000.56	Technician	23-02-1975
E5	Roy	24	34000.67	Programmer	05-05-1990
E6	Sindhu	33	36000.45	Technician	09-04-1984

1. Display the category of workers.
2. Find all employees with salary greater than 35000 who are not programmers.
3. Display employees with 6 digit salary.
4. Find all employees with an experience more than 35 years.
5. Display all employees who are not a Manager or a Programmer.

Prerequisite Knowledge:
NUMERIC FUNCTIONS
ABS: It returns the absolute value of n.
POWER: It returns m raised to nth power. n must be an integer else an error is returned.
ROUND: It returns n rounded to m places right of the decimal point. If m is omitted, n is rounded to zero places. m must be an integer.

SQRT: It returns square root of n. n should be greater than zero.

STRING FUNCTIONS

LOWER: It returns char with letters in lower case.
INITCAP: It returns char with the first letter in upper case.
UPPER: It returns char with all letters forced to upper case.
SUBSTR: It returns a portion of char beginning at character m, exceeding up to n characters. If n is omitted result is written up to the end character. The 1stposition of char is one.
LENGTH: It returns the length of char
LTRIM: It removes characters from the left of char with initial characters removed up to the 1st character not in set.
RTRIM: It returns char with final characters removed after the last character not in the set. Set is optional. It defaults to spaces.
LPAD: It returns char1, left padded to length n with the sequence of characters in char2. char2 defaults to blanks.
RPAD: It returns char1, right padded to length n with the characters in char2, replicated as many times as necessary. If char2 is omitted, it is padded with blanks.

CONVERSION FUCTIONS

TO_ NUMBER(CHAR): It converts the char value containing a number to a value of number data type.
TO_CHAR(N,FMT): It converts a value of number data type to a value of char data type, using the optional format string. It accepts a number n and a numeric format fmt in which the number has to appear. If fmt is omitted, n is converted to a char value exactly long enough to hold significant digits.
TO_CHAR(DATE, FMT): It converts a value of data type to char value. It accepts a date as well as the format in which the date has to appear. Fmt must be a date format. If fmt is omitted, date is the default date format.

DATE FUNCTIONS

SYSDATE : The sysdate is a pseudo column that contains the current date and time. It requires no arguments when selected from the table dual and returns the current date.
ADD_MONTHS(D,N): It returns date after adding the number of months specified with the function.
LAST_DAY(D): It returns the last date of the month specified with the

function.
MONTHS_BETWEEN(D1,D2): It returns number of months between D1 and D2.
NEXT_DAY(DATE, CHAR): It returns the date of the first week day named by char . char must be a day of the week.

Chapter 6

Experiment 3

Experiment:Add the following constraints to the table created in Experiment 2.

1. Set Employee Id as primary key for the Employee table created in Experiment 2.
2. Ensure that the Employee Id is unique.
3. Ensure that Name of the Employee is not left blank while entering employee details.
4. Change the primary key of this table from Employee id to Employee Id and Name

Prerequisite knowledge :

SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.

Constraints can be divided into following two types,

1. **Column level constraints :** limits only column data
2. **Table level constraints :** limits whole table data Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

- NOT NULL
- UNIQUE
- PRIMARY KEY

- FOREIGN KEY
- CHECK
- DEFAULT

1. **NOT NULL Constraint**

   NOT NULL constraint restricts a column from having a NULL value. Once NOT NULL constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value. One important point to note about NOT NULL constraint is that it cannot be defined at table level.

   Example using NOT NULL constraint

   CREATE table Student(sid int NOT NULL, Name varchar(60), Age int);
   The above query will declare that the sid field of Student table will not take NULL value.

2. **UNIQUE Constraint**

   UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data. UNIQUE constraint can be applied at column level or table level.

   Example using UNIQUE constraint when creating a Table (Table Level)

   CREATE table Student(sid int NOT NULL UNIQUE, Name varchar(60), Age int);
   The above query will declare that the sid field of Student table will only have unique values and wont take NULL value.

   Example using UNIQUE constraint after Table is created (Column Level)

   ALTER table Student add UNIQUE(sid);
   The above query specifies that sid field of Student table will only have unique value.

3. **Primary Key Constraint**

   Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

   Example using PRIMARY KEY constraint at Table Level

CREATE table Student (sid int PRIMARY KEY, Name varchar(60) NOT NULL, Age int);
The above command will creates a PRIMARY KEY on the sid.

Example using PRIMARY KEY constraint at Column Level

ALTER table Student add PRIMARY KEY (sid);
The above command will creates a PRIMARY KEY on the sid.

4. **Foreign Key Constraint**

   FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables.

Chapter 7

Experiment 4

Table 1:emp - The table should contain the following fields

1. Empid
2. Deptid
3. Job
4. Salary

Table 2:emp.details - The table should contain the following fields

1. Empid
2. Fname
3. Lname
4. City
5. Hiredate

Table 3:dept.details - The table should contain the following fields

1. Deptid
2. Title
3. Location

Give suitable attributes and constraint to the fields.

Perform the following retrieval operations.

1. Write an SQL query to display all the employees that have the same number of letters in the first name and last name.
2. Write an SQL query that will display the name of cities with first letter capital letter and everything else lowercase.
3. Write an SQL query to display the Empid, first name and last name of employees whose id numbers are 4,3,5 and 1. List them in alphabetic order.
4. Write an SQL query that displays the total number of records in the emp table.
5. Write an SQL query to list the first name, last name and deptid of those in deptid 3 who are either a programmer or a project leader.
6. Write an SQL query to list the first name, last name, dept id and salary of those whose salary i between 7000 and 15000.
7. Write an SQL query to raise the salary for all the employees in dept id 4 and 2 by 15 percentage.
8. Count the total number of rows in the employee table excluding NULL values.
9. List out the sysdate and last date of that month.
10. Display the squared values of employee ids.
11. Display the contents of the records in the employee table, those works in the sales department in th following format. Mr/Mrs.....is working at.....department of xyz company, located at .....
12. List out the details of employees who has got more than one year working experience in the present working firm.
13. Find out the next day of Thursday from sysdate.
14. Convert the names of department into lower to upper case.
15. Write an SQL query to display the employees hire date in the format 'mm/dd/yyyy'.

16. Find out the average salary paid to the employees in he grouped order of each department.

17. Write an SQL query that displays the details of employees with odd numbered employee id. Use mod function.

18. Write an SQL query that displays maximum and minimum salary for each department, where the minimum salary is less than 5000.

19. One employee was hired 15th January ,1996 and another employee hired on 16th September 1997. Write an SQL query to show how many months between their hiring.

20. Write an SQL query to display different job categories from emp table.

21. Write an SQL query to display the name and hiring date for the sales managers. Put them in as sentence format to look "Catherine John was hired on 30th of June 2004.

22. Create a save-point named SP.

23. Add a new column named Date of birth with character type to table employee details. Insert values for the new column in the format dd/mm/yy and calculate age of each employee.

24. Write an SQL query to display the total number of employees in each department, where there are more than 10 employees.

25. Rollback he transaction up to save-point SP.

26. Truncate the department details.

27. Write an SQL query to create a new user and perform the following actions. a)Change password b) Grant Privilege c) Revoke Privilege

28. Write an SQL query to create a view of empid and job from emp table and alter the name of view.

29. Drop the created view.

Chapter 8

Experiment 5

Experiment: Create the following tables without using an insert operation for already existing data.

Employee Table:

Employee Id	Name	Age	Title	Joining Date
E1	Mahesh	50	Manager	20-03-1967
E2	Nimitha	35	Programmer	02-08-1981
E3	Ashitha	28	Programmer	01-09-1988
E4	Ajay	32	Technician	23-02-1975
E5	Roy	24	Programmer	05-05-1990
E6	Sindhu	33	Technician	09-04-1984

EmpDepartment Table:

Employee Id	Joining Date	Department
E1	20-03-1967	D4
E2	02-08-1981	D2
E3	01-09-1988	D1
E4	23-02-1975	D1
E5	05-05-1990	D1
E6	09-04-1984	D2

EmpTitle Table:

Employee Id	Title
E1	Manager
E2	Programmer
E3	Programmer
E4	Technician
E5	Programmer
E6	Technician

EmpSalary Table:

Employee Id	Salary
E1	115000.89
E2	110000.78
E3	35000.89
E4	27000.56
E5	34000.67
E6	36000.45

Department Table:

Department Id	Department
D1	Production
D2	R and D
D3	Purchasing
D4	Marketing

Prerequisite Knowledge:
SQL SELECT INTO

- The SELECT INTO statement copies data from one table into a new table.

  SELECT *INTO newtable [IN externaldb]FROM oldtable WHERE condition;

- The following query Copy only some columns into a new table:

  SELECT column1, column2, column3, ... INTO newtable [IN externaldb] FROM oldtable WHERE condition;

- The new table will be created with the column-names and types as defined in the old table. You can create new column names using the AS clause.

SQL SELECT INTO Examples
The following SQL statement creates a backup copy of Customers:

*SELECT * INTO CustomersBackup FROM Customers;*
The following SQL statement uses the IN clause to copy the table into a new table in another database:

*SELECT * INTO CustomersBackup2017 IN 'Backup.mdb' FROM Customers;*
The following SQL statement copies only a few columns into a new table:

SELECT CustomerName, ContactName INTO CustomersBackup2017 FROM Customers;
The following SQL statement copies only the German customers into a new table:

*SELECT * INTO CustomersGermany FROM Customers WHERE Country = 'Germany';*
The following SQL statement copies data from more than one table into a new table:

SELECT Customers.CustomerName, Orders.OrderID INTO CustomersOrderBackup2017 FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

SQL ALTER TABLE Statement

ALTER TABLE
The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

- To add a column in a table, use the following syntax:

ALTER TABLE tablename ADD columnname datatype;

- To delete a column in a table, use the following syntax

  ALTER TABLE tablename DROP COLUMN columnname;

- To change the data type of a column in a table, use the following syntax:
  ALTER TABLE tablename ALTER COLUMN columnname datatype;
  ALTER TABLE tablename MODIFY COLUMN columnname datatype;
  or
  ALTER TABLE tablename MODIFY columnname datatype;

The SQL UPDATE Statement
The UPDATE statement is used to modify the existing records in a table.

- UPDATE Syntax: UPDATE tablename SET column1 = value1, column2 = value2, ... WHERE condition;
- UPDATE Multiple Records: It is the WHERE clause that determines how many records that will be updated.

  The following SQL statement will update the contactname to "Juan" for all records where country is "Mexico":

  Example *UPDATE Customers SET ContactName='Juan' WHERE Country='Mexico';*
  Change Data Type Example
- Now we want to change the data type of the column named "DateOfBirth" in the "Persons" table.

  We use the following SQL statement:

  ALTER TABLE Persons ALTER COLUMN DateOfBirth year;
- Next, we want to delete the column named "DateOfBirth" in the "Persons" table.

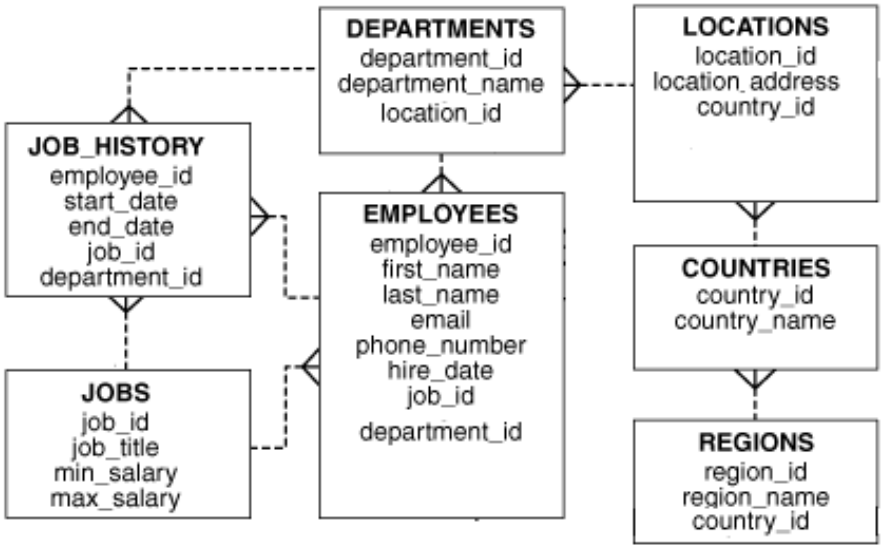
  We use the following SQL statement:

  ALTER TABLE Persons DROP COLUMN DateOfBirth;

Chapter 9

Experiment 6

Experiment: Establish the relation shown in the following schema diagram.



Prerequisite Knowledge:
The relations between tables can be created using Primary key Foriegn Key Concepts. There are basically three types of relations:

- One-One Relationship (1-1 Relationship)
- One-Many Relationship (1-M Relationship)
- Many-Many Relationship (M-M Relationship)

Experiment:
Employee Table

EId	FName	LName	email	Phone	HireDate	JId	DId
E1	Mini	P.K	mini@gmail.com	9955	12-04-1987	J1	D1
E2	Savanth	Kumar	savanth@gmail.com	9867	22-07-1990	J2	D2
E3	Jane	Joy	jane@gmail.com	9795	23-04-1999	J2	D3
E4	Suhail	Haroon	suhail@gmail.com	9234	01-05-2003	J3	D3
E5	Radha	Soman	radha@gmail.com	9976	09-02-2011	J2	D1

Deaprtment Table

DId	DName	Location Id
D1	Production	L1
D2	Marketing	L1
D3	RandD	L1
D1	Marketing	L2
D2	HR	L2
D1	Production	L3
D2	RandD	L3
D3	HR	L3

Job History Table

EId	StartDate	End Date	Jid	Did
E1	12-04-1987	02-05-1993	J1	D1
E1	02-05-1993	30-03-2008	J2	D1
E1	30-03-2008	-	J3	D1
E2	22-07-1990	05-06-2001	J7	D2
E2	05-06-2001	-	J8	D3
E3	23-04-1999	09-11-2004	J5	D1
E3	09-11-2004	-	J6	D3
E4	01-05-2003	-	J7	D3

Jobs Table

JId	Title	Min-Salary	Max-Salary
J1	Junior Programmer	24000	45000
J2	Senior Programmer	45000	112000
J3	Project Manager	67000	135000
J4	Junior Researcher	35000	67000
J5	Senior Researcher	65000	132000
J6	Scientist	78000	178000
J7	Manager	32000	56000
J8	Marketing Head	21000	45000
J9	HR Head	34000	45000

Region Table

Region Id	Region Name	Country
R1	North East	C1
R2	North India	C2
R33	South India	C2

Location Table

LId	L Address	Country Id
L1	New York	C1
L2	Delhi	C2
L3	Kochi	C2

Country Table

Country Id	Country Name
C1	India
C2	USA

Chapter 10

Join Operation

To retrieve data from a database that has relationships, we often need to use JOIN queries.

Prerequisite Knowledge:

Different Types of SQL JOINs

1. (INNER) JOIN: Returns records that have matching values in both tables
2. LEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the right table
3. RIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the left table
4. FULL (OUTER) JOIN: Return all records when there is a match in either left or right table

Experiment 7

Perform the following retrieval operation from the given tables.
Book List

BID	Title	Author	Status
101	Fasting Feasting	Anitha Desai	Issued
102	Untouchable	MulkrajAnand	Not Issued
103	Zahir	Paulo Coelo	Not Issued
104	Ancient Promisesr	Jayasree Misra	Issued
105	I Dare	Kiran Bedi	Issued

Book Issued

BID	UID	Issue Date
101	32	12-june-05
104	67	30-June-05
105	67	10-July-05

User List

UID	Name	Course	Semester	NOB
32	Flozy	ECE	3	10
33	Selina	ECE	4	0
34	Ajith	CS	4	0
67	Grace	IT	7	2

1. Find out the details of the students who took the book "Fasting Feasting"
2. Find out the details of the books issued before 15th June.
3. Find out the details of the 4th Semester Students who took the book at least once
4. Display the names of the book issued on date 12th June 2005.
5. Display the names of book issued most recently
6. Display the name of the student who has taken book on month July.
7. Display the name of students who took the book written by Kiran Bedi.
8. Display the name of the books issued to the user Flozy.

Chapter 11

PL/SQL

PL/SQL (Procedural Language/Structured Query Language) is Oracle Corporation's procedural extension for SQL and the Oracle relational database. PL/SQL includes procedural language elements such as conditions and loops. It allows declaration of constants and variables, procedures and functions, types and variables of those types, and triggers. It can handle exceptions (runtime errors). Arrays are supported involving the use of PL/SQL collections. Implementations from version 8 of Oracle Database onwards have included features associated with object-orientation. One can create PL/SQL units such as procedures, functions, packages, types, and triggers, which are stored in the database for reuse by applications that use any of the Oracle Database programmatic interfaces.

PL/SQL - Basic Syntax

1. Declarations

   This section starts with the keyword DECLARE. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.

2. Executable Commands

   This section is enclosed between the keywords BEGIN and END and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a NULL command to indicate that nothing should be executed.

3. Exception Handling

This section starts with the keyword EXCEPTION. This optional section contains exception(s) that handle errors in the program.

Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using BEGIN and END. Following is the basic structure of a PL/SQL block DECLARE

```
DECLARE
   <declarations section>
BEGIN
   <executable command(s)>
EXCEPTION
   <exception handling>
END;
```

The 'Hello World' Example

```
DECLARE
   message  varchar2(20):= 'Hello, World!';
BEGIN
   dbms_output.put_line(message);
END;
/
```

The PL/SQL Identifiers PL/SQL identifiers are constants, variables, exceptions, procedures, cursors, and reserved words. The identifiers consist of a letter optionally followed by more letters, numerals, dollar signs, underscores, and number signs and should not exceed 30 characters.

By default, identifiers are not case-sensitive. So you can use integer or INTEGER to represent a numeric value. You cannot use a reserved keyword as an identifier.

The PL/SQL Delimiters A delimiter is a symbol with a special meaning. Following is the list of delimiters in PL/SQL

Delimiter	Description
+, -, *, /	Addition, subtraction/negation, multiplication, division
%	Attribute indicator
'	Character string delimiter
.	Component selector
(,)	Expression or list delimiter
:	Host variable indicator
,	Item separator
"	Quoted identifier delimiter
=	Relational operator
@	Remote access indicator
;	Statement terminator
:=	Assignment operator
=>	Association operator
||	Concatenation operator
**	Exponentiation operator
<<, >>	Label delimiter (begin and end)
/*, */	Multi-line comment delimiter (begin and end)
--	Single-line comment indicator
..	Range operator
<, >, <=, >=	Relational operators
<>, '=, ~=, ^=	Different versions of NOT EQUAL

PL/SQL Program Units A PL/SQL unit is any one of the following

- PL/SQL block
- Function
- Package
- Package body
- Procedure
- Trigger
- Type
- Type body

PL/SQL - Data Types The PL/SQL variables, constants and parameters must have a valid data type, which specifies a storage format, constraints, and a valid range of values.

S.No	Category & Description
1	**Scalar** Single values with no internal components, such as a **NUMBER, DATE,** or **BOOLEAN**.
2	**Large Object (LOB)** Pointers to large objects that are stored separately from other data items, such as text, graphic images, video clips, and sound waveforms.
3	**Composite** Data items that have internal components that can be accessed individually. For example, collections and records.
4	**Reference** Pointers to other data items.

PL/SQL Scalar Data Types and Subtypes

S.No	Category & Description
1	**Scalar** Single values with no internal components, such as a **NUMBER, DATE,** or **BOOLEAN**.
2	**Large Object (LOB)** Pointers to large objects that are stored separately from other data items, such as text, graphic images, video clips, and sound waveforms.
3	**Composite** Data items that have internal components that can be accessed individually. For example, collections and records.
4	**Reference** Pointers to other data items.

PL/SQL Numeric Data Types and Subtypes

1. **PLS_INTEGER**

   Signed integer in range -2,147,483,648 through 2,147,483,647, represented in 32 bits

2. **BINARY_INTEGER**

   Signed integer in range -2,147,483,648 through 2,147,483,647, represented in 32 bits

3. **BINARY_FLOAT**

   Single-precision IEEE 754-format floating-point number

4. **BINARY_DOUBLE**

   Double-precision IEEE 754-format floating-point number

5. **NUMBER(prec, scale)**

   Fixed-point or floating-point number with absolute value in range 1E-130 to (but not including) 1.0E126. A NUMBER variable can also represent 0

6. **DEC(prec, scale)**

   ANSI specific fixed-point type with maximum precision of 38 decimal digits

7. **DECIMAL(prec, scale)**

   IBM specific fixed-point type with maximum precision of 38 decimal digits

8. **NUMERIC(pre, secale)**

   Floating type with maximum precision of 38 decimal digits

9. **DOUBLE PRECISION**

   ANSI specific floating-point type with maximum precision of 126 binary digits (approximately 38 decimal digits)

10. **FLOAT**

    ANSI and IBM specific floating-point type with maximum precision of 126 binary digits (approximately 38 decimal digits)

11. **INT**

    ANSI specific integer type with maximum precision of 38 decimal digits

12. **INTEGER**

    ANSI and IBM specific integer type with maximum precision of 38 decimal digits

13. **SMALLINT**

    ANSI and IBM specific integer type with maximum precision of 38 decimal digits

14. **REAL**

    Floating-point type with maximum precision of 63 binary digits (approximately 18 decimal digits)

```
DECLARE
   num1 INTEGER;
   num2 REAL;
   num3 DOUBLE PRECISION;
BEGIN
   null;
END;
/
```

PL/SQL Character Data Types and Subtypes

1. CHAR

   Fixed-length character string with maximum size of 32,767 bytes

2. VARCHAR2

   Variable-length character string with maximum size of 32,767 bytes

3. RAW

   Variable-length binary or byte string with maximum size of 32,767 bytes, not interpreted by PL/SQL

4. NCHAR

   Fixed-length national character string with maximum size of 32,767 bytes

5. NVARCHAR2

   Variable-length national character string with maximum size of 32,767 bytes

6. LONG

   Variable-length character string with maximum size of 32,760 bytes

7. LONG RAW

   Variable-length binary or byte string with maximum size of 32,760 bytes, not interpreted by PL/SQL

8. ROWID

   Physical row identifier, the address of a row in an ordinary table

9. UROWID

   Universal row identifier (physical, logical, or foreign row identifier)

PL/SQL Character Data Types and Subtypes

1. CHAR

   Fixed-length character string with maximum size of 32,767 bytes

2. VARCHAR2

   Variable-length character string with maximum size of 32,767 bytes

3. RAW

   Variable-length binary or byte string with maximum size of 32,767 bytes, not interpreted by PL/SQL

4. NCHAR

   Fixed-length national character string with maximum size of 32,767 bytes

5. NVARCHAR2

   Variable-length national character string with maximum size of 32,767 bytes

6. LONG

   Variable-length character string with maximum size of 32,760 bytes

7. LONG RAW

   Variable-length binary or byte string with maximum size of 32,760 bytes, not interpreted by PL/SQL

8. ROWID

   Physical row identifier, the address of a row in an ordinary table

9. UROWID

   Universal row identifier (physical, logical, or foreign row identifier)

PL/SQL Datetime and Interval Types The DATE datatype is used to store fixed-length datetimes, which include the time of day in seconds since midnight. Valid dates range from January 1, 4712 BC to December 31, 9999 AD.

The default date format is set by the Oracle initialization parameter . For example, the default might be 'DD-MON-YY', which includes a two-digit number for the day of the month, an abbreviation of the month name, and the last two digits of the year. For example, 01-OCT-12.

Field Name	Valid Datetime Values	Valid Interval Values
YEAR	-4712 to 9999 (excluding year 0)	Any nonzero integer
MONTH	01 to 12	0 to 11
DAY	01 to 31 (limited by the values of MONTH and YEAR, according to the rules of the calendar for the locale)	Any nonzero integer
HOUR	00 to 23	0 to 23
MINUTE	00 to 59	0 to 59
SECOND	00 to 59.9(n), where 9(n) is the precision of time fractional seconds	0 to 59.9(n), where 9(n) is the precision of interval fractional seconds
TIMEZONE_HOUR	-12 to 14 (range accommodates daylight savings time changes)	Not applicable
TIMEZONE_MINUTE	00 to 59	Not applicable
TIMEZONE_REGION	Found in the dynamic performance view V$TIMEZONE_NAMES	Not applicable
TIMEZONE_ABBR	Found in the dynamic performance view V$TIMEZONE_NAMES	Not applicable

Variable Declaration in PL/SQL PL/SQL variables must be declared in the declaration section or in a package as a global variable. When you declare a variable, PL/SQL allocates memory for the variable's value and the storage location is identified by the variable name.

The syntax for declaring a variable is

```
variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT initial_value]
```

Where, variable name is a valid identifier in PL/SQL, datatype must be a valid PL/SQL data type or any user defined data type which we already have discussed in the last chapter. Some valid variable declarations along with their definition are shown below

```
sales number(10, 2);
pi CONSTANT double precision := 3.1415;
name varchar2(25);
address varchar2(100);
```

Initializing Variables in PL/SQL Whenever you declare a variable, PL/SQL assigns it a default value of NULL. If you want to initialize a variable with a value other than the NULL value, you can do so during the declaration, using either of the following

-The DEFAULT keyword

-The assignment operator

For example

```
counter binary_integer := 0;
greetings varchar2(20) DEFAULT 'Have a Good Day';
```

Eg Program:

```
DECLARE
   a integer := 10;
   b integer := 20;
   c integer;
   f real;
BEGIN
   c := a + b;
   dbms_output.put_line('Value of c: ' || c);
   f := 70.0/3.0;
   dbms_output.put_line('Value of f: ' || f);
END;
/
```

Assigning SQL Query Results to PL/SQL Variables You can use the SELECT INTO statement of SQL to assign values to PL/SQL variables. For each item in the SELECT list, there must be a corresponding, type-compatible variable in the INTO list.

```
DECLARE
   c_id customers.id%type := 1;
   c_name  customerS.No.ame%type;
   c_addr customers.address%type;
   c_sal  customers.salary%type;
BEGIN
   SELECT name, address, salary INTO c_name, c_addr, c_sal
   FROM customers
   WHERE id = c_id;
   dbms_output.put_line
   ('Customer ' ||c_name || ' from ' || c_addr || ' earns ' || c_sal);
END;
/
```

PL/SQL - Conditions

1. IF - THEN statement The IF statement associates a condition with a sequence of statements enclosed by the keywords THEN and END IF.

If the condition is true, the statements get executed and if the condition is false or NULL then the IF statement does nothing.

2. IF-THEN-ELSE statement IF statement adds the keyword ELSE followed by an alternative sequence of statement. If the condition is false or NULL, then only the alternative sequence of statements get executed. It ensures that either of the sequence of statements is executed.

3. IF-THEN-ELSIF statement It allows you to choose between several alternatives.

4. Case statement Like the IF statement, the CASE statement selects one sequence of statements to execute.

   However, to select the sequence, the CASE statement uses a selector rather than multiple Boolean expressions. A selector is an expression whose value is used to select one of several alternatives.

5. Searched CASE statement The searched CASE statement has no selector, and it's WHEN clauses contain search conditions that yield Boolean values.

6. nested IF-THEN-ELSE You can use one IF-THEN or IF-THEN-ELSIF statement inside another IF-THEN or IF-THEN-ELSIF statement(s).

Chapter 12

Experiment 8

Reverse of a String

Write a PL/SQL program to find the reverse of a string

```
REVERSE OF A STRING
declare
    str1 varchar2(50):='&str';
    str2 varchar2(50);
    len number;
    i number;

begin
    len:=length(str1);

    for i in reverse 1..len
    loop
        str2:=str2 || substr(str1,i,1);
    end loop;

    dbms_output.put_line('Reverse of String is:'||str2);
end;
```

Chapter 13

PL/SQL - Loops

PL/SQL - Loops

1. PL/SQL Basic LOOP In this loop structure, sequence of statements is enclosed between the LOOP and the END LOOP statements. At each iteration, the sequence of statements is executed and then control resumes at the top of the loop.

2. PL/SQL WHILE LOOP Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

3. PL/SQL FOR LOOP Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

4. Nested loops in PL/SQL You can use one or more loop inside any another basic loop, while, or for loop.

The Loop Control Statements

1. EXIT statement The Exit statement completes the loop and control passes to the statement immediately after the END LOOP.

2. CONTINUE statement Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

3. GOTO statement Transfers control to the labeled statement. Though it is not advised to use the GOTO statement in your program.

Chapter 14

PL/SQL - Strings

PL/SQL offers three kinds of strings

Fixed-length strings In such strings, programmers specify the length while declaring the string. The string is right-padded with spaces to the length so specified.

Variable-length strings In such strings, a maximum length up to 32,767, for the string is specified and no padding takes place.

Character large objects (CLOBs) These are variable-length strings that can be up to 128 terabytes. Oracle database provides numerous string datatypes, such as CHAR, NCHAR, VARCHAR2, NVARCHAR2, CLOB, and NCLOB.

PL/SQL String Functions and Operators

1. ASCII(x);

   Returns the ASCII value of the character x.

2. CHR(x);

   Returns the character with the ASCII value of x.

3. CONCAT(x, y);

   Concatenates the strings x and y and returns the appended string.

4. INITCAP(x);

   Converts the initial letter of each word in x to uppercase and returns that string.

5. INSTR(x, find_string [, start] [, occurrence]);

   Searches for find_string in x and returns the position at which it occurs.

6. INSTRB(x);

   Returns the location of a string within another string, but returns the value in bytes.

7. LENGTH(x);

   Returns the number of characters in x.

8. LENGTHB(x);

   Returns the length of a character string in bytes for single byte character set.

9. LOWER(x);

   Converts the letters in x to lowercase and returns that string.

10. LPAD(x, width [, pad_string]) ;

    Pads x with spaces to the left, to bring the total length of the string up to width characters.

11. LTRIM(x [, trim_string]);

    Trims characters from the left of x.

12. NANVL(x, value);

    Returns value if x matches the NaN special value (not a number), otherwise x is returned.

13. NLS_INITCAP(x);

    Same as the INITCAP function except that it can use a different sort method as specified by NLSSORT.

14. NLS_LOWER(x) ;

    Same as the LOWER function except that it can use a different sort method as specified by NLSSORT.

15. NLS_UPPER(x);

    Same as the UPPER function except that it can use a different sort method as specified by NLSSORT.

16. NLSSORT(x);

    Changes the method of sorting the characters. Must be specified before any NLS function; otherwise, the default sort will be used.

17. NVL(x, value);

    Returns value if x is null; otherwise, x is returned.

18. NVL2(x, value1, value2);

    Returns value1 if x is not null; if x is null, value2 is returned.

19. REPLACE(x, search_string, replace_string);

    Searches x for search_string and replaces it with replace_string.

20. RPAD(x, width [, pad_string]);

    Pads x to the right.

21. RTRIM(x [, trim_string]);

    Trims x from the right.

22. SOUNDEX(x) ;

    Returns a string containing the phonetic representation of x.

23. SUBSTR(x, start [, length]);

    Returns a substring of x that begins at the position specified by start. An optional length for the substring may be supplied.

24. SUBSTRB(x);

    Same as SUBSTR except that the parameters are expressed in bytes instead of characters for the single-byte character systems.

25. TRIM([trim_char FROM) x);

    Trims characters from the left and right of x.

26. UPPER(x);

    Converts the letters in x to uppercase and returns that string.

Chapter 15

Experiment 9

Experiment: Declare three strings "John Smith", "Infotech" and "InfoTech has been serving over 300 clients, both nationwide and worldwide including North America, Europe and Australia. Due to the experience, knowledge base, technology diversity and market diversity that we have attained over the years, both in planning and executing information technologies, we offer our clients a unique service that cannot be offered by many other companies.

InfoTech has systematically received the highest awards and recognitions from some of the most demanding customers in the market, including the U.S. Navy, Raytheon Missile Systems, UBS, Wells Fargo, the State of North Dakota, SPAWAR, ISI (International Strategy and Investment), Raymond James and GMI Ratings, to name just a few.

InfoTechs offices in Europe and throughout the country have allowed young professionals and recent graduates in the high tech field the opportunity to build careers in their home states cities. InfoTech was named one of the top ten companies to work for and one of the top five companies to intern for in North Dakota."

```
DECLARE
   name varchar2(20);
   company varchar2(30);
   introduction clob;
   choice char(1);
BEGIN
   name := 'John Smith';
   company := 'Infotech';
   introduction := ' Hello! I''m John Smith from Infotech.';
   choice := 'y';
   IF choice = 'y' THEN
      dbms_output.put_line(name);
      dbms_output.put_line(company);
      dbms_output.put_line(introduction);
   END IF;
END;
/
```

Chapter 16

Arrays in PL/SQL

PL/SQL - Arrays The PL/SQL programming language provides a data structure called the VARRAY, which can store a fixed-size sequential collection of elements of the same type. A varray is used to store an ordered collection of data, however it is often better to think of an array as a collection of variables of the same type. A varray type is created with the CREATE TYPE statement. You must specify the maximum size and the type of elements stored in the varray.

The basic syntax for creating a VARRAY type at the schema level is

```
CREATE OR REPLACE TYPE varray_type_name IS VARRAY(n) of <element_type>
```

Where,

- *varray_type_name* is a valid attribute name,
- *n* is the number of elements (maximum) in the varray,
- *element_type* is the data type of the elements of the array.

Chapter 17

Experiment 10-13

Experiment 10 : Create two arrays ,one for keeping names and one for keeping marks. Display total count of students and list of students and their marks.

```
DECLARE
   type namesarray IS VARRAY(5) OF VARCHAR2(10);
   type grades IS VARRAY(5) OF INTEGER;
   names namesarray;
   marks grades;
   total integer;
BEGIN
   names := namesarray('Kavita', 'Pritam', 'Ayan', 'Rishav', 'Aziz');
   marks:= grades(98, 97, 78, 87, 92);
   total := names.count;
   dbms_output.put_line('Total '|| total || ' Students');
   FOR i in 1 .. total LOOP
      dbms_output.put_line('Student: ' || names(i) || '
      Marks: ' || marks(i));
   END LOOP;
END;
/
```

Experiment 11
Write a PL/SQL program to compute volume of a cuboid.
Experiment 12
Write a PL/SQL program to find the prime numbers from 2 to 100
Experiment 13
Write a PL/SQL program to find the factorial of a number

Chapter 18

PL/SQL - Implicit Cursors

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors

1. Implicit cursors

2. Explicit cursors

Implicit Cursors Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

S.No	Attribute & Description
1	**%FOUND** Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
2	**%NOTFOUND** The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
3	**%ISOPEN** Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	**%ROWCOUNT** Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Chapter 19

Experiment 14 Write a program that will update the table and increase the salary of each customer by 500 and use the SQL

```
DECLARE
   total_rows number(2);
BEGIN
   UPDATE customers
   SET salary = salary + 500;
   IF sql%notfound THEN
      dbms_output.put_line('no customers selected');
   ELSIF sql%found THEN
      total_rows := sql%rowcount;
      dbms_output.put_line( total_rows || ' customers selected ');
   END IF;
END;
/
```

Chapter 20

PL/SQL - Explicit Cursors

Explicit Cursors Explicit cursors are programmer-defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is

```
CURSOR cursor_name IS select_statement;
```

Working with an explicit cursor includes the following steps

1. Declaring the cursor for initializing the memory
2. Opening the cursor for allocating the memory
3. Fetching the cursor for retrieving the data
4. Closing the cursor to release the allocated memory

Declaring the Cursor Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example

```
CURSOR c_customers IS
   SELECT id, name, address FROM customers;
```

Opening the Cursor Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it.

```
OPEN c_customers;
```

Fetching the Cursor Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the Cursor Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows

```
CLOSE c_customers;
```

20.1 Experiment 15

Display the name of customers from the customer table.

```
DECLARE
   c_id customers.id%type;
   c_name customerS.No.ame%type;
   c_addr customers.address%type;
   CURSOR c_customers is
      SELECT id, name, address FROM customers;
BEGIN
   OPEN c_customers;
   LOOP
   FETCH c_customers into c_id, c_name, c_addr;
      EXIT WHEN c_customers%notfound;
      dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
   END LOOP;
   CLOSE c_customers;
END;
/
```

20.2 Experiment 16

1. Write a PL/SQL script to update the salary table as follows.
   - Increment salary by rs 1000 for id=1
   - Increment salary by rs 2000 for id=2
   - Increment salary by rs 5000 for id=3
2. Create the following tables with the fields specified.
   - jobs(jcode,jname,qualification,salaryoffered,jplace, noofvacancies)

- applicants(appcode, appname, qualification, expected salary, place-chosen, placedornot)
- placelist(plcode,appcode,jcode)

20.3 Experiment 17

Write a PL/SQL script to select best suitable job for applicants and store it in a placelist.

- Selection criteria are vacancy available, qualification, should match and salary offered > expected salary.
- Place chosen and jplace should match unless place chosen= NIL.
- Update the number of vacancies and status of placed or not while selection.
- Display details of person/ persons who get maximum salary.

```
DECLARE
CURSOR C1 IS SELECT
APPCODE,APPNAME,QUALIFICATION,EXPECSALARY,PLACE,PLACED FROM APPLICANTS;
CUROSR C2 IS SELECT JCODE, JNAME, QUALI, SALARY, JPLACE, AND VACCANCY FROM JOBS;
AC NUMBER (6);
AN VARCHAR (20);
Q VARCHAR (20);
ES NUMBER (10);
PL VARCHAR (20);
PLACED NUMBER (5):=1;
JC NUMBER (5);
JN VARCHAR (20);
QF VARCHAR (20);
SALARY NUMBER (6);
JP VARCHAR (20);
VAC NUMBER (5);
```

```
BEGIN
OPEN C1;
LOOP
FETCH C1 INTO AC,AN,Q,ES,PL,PLC;
EXIT WHEN C1%NOT FOUND
OPEN C2;
LOOP
FETCH C2 INTO JC,JN,QF,SALARY,JP,VAC;
EXIT WHEN C2%NOT FOUND ;
IF VAC>0 AND QF=Q AND SALARY> ES AND (PL=JP OR PL='NIL')
THEN INSERT INTO PLACED VALUES (PLC,AC,JC);
PLC: = PLC+1;
UPDATE JOBS SET VAC=VAC-1 WHERE JCODE= JC;
UPDATE APPLICANTS SET PLACED ='PLACED' WHERE APPCODE=AC;
END IF;
END LOOP;
CLOSE C2;
END LOOP;
CLOSE C1;
END;
```

Chapter 21

PL/SQL - Trigger

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating Triggers The syntax for creating a trigger is

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Where,

- CREATE [OR REPLACE] TRIGGER trigger name Creates or replaces an existing trigger with the trigger name.
- (BEFORE or AFTER or INSTEAD OF) This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- (INSERT [OR] UPDATE [OR] DELETE)This specifies the DML operation.
- (OF colname) This specifies the column name that will be updated.
- (ON tablename) This specifies the name of the table associated with the trigger.
- (REFERENCING OLD AS o NEW AS n) This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- (FOR EACH ROW) This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

  NOTE:

You will not be able to create triggers for the tables you have created using sys user. Sys is the default user. Permission is not given to create the trigger for user sys due to security reason. So you have to create a new user and grant permissions to this user. After creating the user all the tables under this user will be named and referenced as username.tablename.

The steps are as follows:

1. Creating a User

   Once connected as SYSTEM, simply issue the CREATE USER command to generate a new account.

   CREATE USER manager IDENTIFIED BY 12345;
   Here were simply creating a manager account that is IDENTIFIED or authenticated by the specified password:12345.

2. Providing Roles

   Typically, youll first want to assign privileges to the user through attaching the account to various roles, starting with the CONNECT role:

   GRANT CONNECT TO manager;

   In some cases to create a more powerful user, you may also consider adding the RESOURCE role (allowing the user to create named types for custom schemas) or even the DBA role, which allows the user to not only create custom named types but alter and destroy them as well.

   GRANT CONNECT, RESOURCE, DBA TO manager;

3. Assigning Privileges

   Next youll want to ensure the user has privileges to actually connect to the database and create a session using GRANT CREATE SESSION. Well also combine that with all privileges using GRANT ANY PRIVILEGES.

   GRANT CREATE SESSION GRANT ANY PRIVILEGE TO manager;

   We also need to ensure our new user has disk space allocated in the system to actually create or modify tables and data, so well GRANT TABLESPACE like so:

   GRANT UNLIMITED TABLESPACE TO manager;

Chapter 22

Experiment 18

Create a trigger for the Employee table that would fire

- When a new Employee joins the company
- When an increment is given to the salary of an employee.
- When an Employee leaves the company.

This trigger should display the salary difference between the old values and new values. (First create a Employee table with fields (id,name,age,address,salary)

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

Triggering a Trigger Perform some DML operation and check trigger.

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00 );
```

Experiment 19:

Create a purchase table with fields

- (purchase id, item,no:of item,rate of item, price)

and a product table which shows number of items in stock with the following fields.

- (item, number of items)

Create a trigger to update the number of items in the product table when a purchase occur.